

讲堂 > Linux性能优化实战 > 文章详情

## 04 | 基础篇：经常说的 CPU 上下文切换是什么意思？（下）

2018-11-28 倪朋飞



### 04 | 基础篇：经常说的 CPU 上下文切换是什么意思？（下）

朗读人：冯永吉 11'56" | 5.47M

你好，我是倪朋飞。

上一节，我给你讲了 CPU 上下文切换的工作原理。简单回顾一下，CPU 上下文切换是保证 Linux 系统正常工作的一个核心功能，按照不同场景，可以分为进程上下文切换、线程上下文切换和中断上下文切换。具体的概念和区别，你也要在脑海中过一遍，忘了的话及时查看上一篇。

今天我们就接着来看，究竟怎么分析 CPU 上下文切换的问题。

### 怎么查看系统的上下文切换情况


通过前面学习我们知道，过多的上下文切换，会把 CPU 时间消耗在寄存器、内核栈以及虚拟内存等数据的保存和恢复上，缩短进程真正运行的时间，成了系统性能大幅下降的一个元凶。

既然上下文切换对系统性能影响那么大，你肯定迫不及待想知道，到底要怎么查看上下文切换呢？在这里，我们可以使用 `vmstat` 这个工具，来查询系统的上下文切换情况。

vmstat 是一个常用的系统性能分析工具，主要用来分析系统的内存使用情况，也常用来分析 CPU 上下文切换和中断的次数。

比如，下面就是一个 vmstat 的使用示例：

```
1 # 每隔 5 秒输出 1 组数据
2 $ vmstat 5
3 procs -----memory----- ---swap-- -----io----- -system-- -----cpu-----
4  r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
5  0  0       0 7005360  91564 818900    0    0     0     0  25  33  0  0 100  0  0
```

 复制代码

我们一起来看看这个结果，你可以先试着自己解读每列的含义。在这里，我重点强调下，需要特别关注的四列内容：


- cs (context switch) 是每秒上下文切换的次数。
- in (interrupt) 则是每秒中断的次数。
- r (Running or Runnable) 是就绪队列的长度，也就是正在运行和等待 CPU 的进程数。
- b (Blocked) 则是处于不可中断睡眠状态的进程数。

可以看到，这个例子中的上下文切换次数 cs 是 33 次，而系统中断次数 in 则是 25 次，而就绪队列长度 r 和不可中断状态进程数 b 都是 0。

vmstat 只给出了系统总体的上下文切换情况，要想查看每个进程的详细情况，就需要使用我们前面提到过的 pidstat 了。给它加上 -w 选项，你就可以查看每个进程上下文切换的情况了。

比如说：

```
1 # 每隔 5 秒输出 1 组数据
2 $ pidstat -w 5
3 Linux 4.15.0 (ubuntu) 09/23/18 _x86_64_ (2 CPU)
4
5 08:18:26      UID      PID   cswch/s nvcschw/s  Command
6 08:18:31        0        1     0.20     0.00  systemd
7 08:18:31        0        8     5.40     0.00  rcu_sched
8 ...
```

 复制代码

这个结果中有两列内容是我们的重点关注对象。一个是 cswch，表示每秒自愿上下文切换 (voluntary context switches) 的次数，另一个则是 nvcschw，表示每秒非自愿上下文切换 (non voluntary context switches) 的次数。

这两个概念你一定要牢牢记住，因为它们意味着不同的性能问题：

- 所谓**自愿上下文切换**，是指进程无法获取所需资源，导致的上下文切换。比如说，I/O、内存等系统资源不足时，就会发生自愿上下文切换。
- 而非**自愿上下文切换**，则是指进程由于时间片已到等原因，被系统强制调度，进而发生的上下文切换。比如说，大量进程都在争抢 CPU 时，就容易发生非自愿上下文切换。

## 案例分析

知道了怎么查看这些指标，另一个问题又来了，上下文切换频率是多少次才算正常呢？别急着要答案，同样的，我们先来看一个上下文切换的案例。通过案例实战演练，你自己就可以分析并找出这个标准了。

## 你的准备

今天的案例，我们将使用 sysbench 来模拟系统多线程调度切换的情况。

sysbench 是一个多线程的基准测试工具，一般用来评估不同系统参数下的数据库负载情况。当然，在这次案例中，我们只把它当成一个异常进程来看，作用是模拟上下文切换过多的问题。

下面的案例基于 Ubuntu 18.04，当然，其他的 Linux 系统同样适用。我使用的案例环境如下所示：


- 机器配置：2 CPU，8GB 内存
- 预先安装 sysbench 和 sysstat 包，如 apt install sysbench sysstat

正式操作开始前，你需要打开三个终端，登录到同一台 Linux 机器中，并安装好上面提到的两个软件包。包的安装，可以先 Google 一下自行解决，如果仍然有问题的，在留言区写下你的情况。

另外注意，下面所有命令，都默认以 **root 用户运行**。所以，如果你是用普通用户登陆的系统，记住先运行 sudo su root 命令切换到 root 用户。

安装完成后，你可以先用 vmstat 看一下空闲系统的上下文切换次数：

```
1 # 间隔 1 秒后输出 1 组数据
2 $ vmstat 1 1
3 procs -----memory----- ---swap-- -----io----- -system-- -----cpu-----
4  r  b   swpd   free   buff  cache   si   so    bi    bo    in    cs us sy id wa st
5  0  0       0 6984064  92668 830896    0    0     2    19    19   35  1  0 99  0  0
```

 复制代码


这里你可以看到，现在的上下文切换次数 cs 是 35，而中断次数 in 是 19，r 和 b 都是 0。因为这会儿我并没有运行其他任务，所以它们就是空闲系统的上下文切换次数。

## 操作和分析

接下来，我们正式进入实战操作。


首先，在第一个终端里运行 `sysbench`，模拟系统多线程调度的瓶颈：

```
1 # 以 10 个线程运行 5 分钟的基准测试，模拟多线程切换的问题
2 $ sysbench --threads=10 --max-time=300 threads run
```

 复制代码

接着，在第二个终端运行 `vmstat`，观察上下文切换情况：

```
1 # 每隔 1 秒输出 1 组数据（需要 Ctrl+C 才结束）
2 $ vmstat 1
3 procs -----memory----- ---swap-- -----io---- -system-- -----cpu-----
4  r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
5  6  0       0 6487428 118240 1292772    0    0    0    0 9019 1398830 16 84  0  0  0
6  8  0       0 6487428 118240 1292772    0    0    0    0 10191 1392312 16 84  0  0  0
```

 复制代码

你应该可以发现，`cs` 列的上下文切换次数从之前的 35 骤然上升到了 139 万。同时，注意观察其他几个指标：

- `r` 列：就绪队列的长度已经到了 8，远远超过了系统 CPU 的个数 2，所以肯定会有大量的 CPU 竞争。
- `us` (`user`) 和 `sy` (`system`) 列：这两列的 CPU 使用率加起来上升到了 100%，其中系统 CPU 使用率，也就是 `sy` 列高达 84%，说明 CPU 主要是被内核占用了。
- `in` 列：中断次数也上升到了 1 万左右，说明中断处理也是个潜在的问题。

综合这几个指标，我们可以知道，系统的就绪队列过长，也就是正在运行和等待 CPU 的进程数过多，导致了大量的上下文切换，而上下文切换又导致了系统 CPU 的占用率升高。

那么到底是什么进程导致了这些问题呢？

我们继续分析，在第三个终端再用 `pidstat` 来看一下，CPU 和进程上下文切换的情况：

```
1 # 每隔 1 秒输出 1 组数据（需要 Ctrl+C 才结束）
2 # -w 参数表示输出进程切换指标，而 -u 参数则表示输出 CPU 使用指标
3 $ pidstat -w -u 1
4 08:06:33      UID      PID    %usr %system  %guest  %wait   %CPU   CPU   Command
5 08:06:34        0    10488    30.00  100.00    0.00    0.00  100.00    0  sysbench
6 08:06:34        0    26326    0.00    1.00    0.00    0.00    1.00    0  kworker/u4:2
7
8 08:06:33      UID      PID  cswch/s nvcsch/s  Command
9 08:06:34        0        8    11.00    0.00  rcu_sched
```

 复制代码

```

10 08:06:34      0      16      1.00      0.00 ksoftirqd/1
11 08:06:34      0     471      1.00      0.00 hv_balloon
12 08:06:34      0    1230      1.00      0.00 iscsid
13 08:06:34      0    4089      1.00      0.00 kworker/1:5
14 08:06:34      0    4333      1.00      0.00 kworker/0:3
15 08:06:34      0   10499      1.00    224.00 pidstat
16 08:06:34      0   26326    236.00      0.00 kworker/u4:2
17 08:06:34   1000   26784    223.00      0.00 sshd

```

从 pidstat 的输出你可以发现，CPU 使用率的升高果然是 sysbench 导致的，它的 CPU 使用率已经达到了 100%。但上下文切换则是来自其他进程，包括非自愿上下文切换频率最高的 pidstat，以及自愿上下文切换频率最高的内核线程 kworker 和 sshd。

不过，细心的你肯定也发现了一个怪异的事儿：pidstat 输出的上下文切换次数，加起来也就几百，比 vmstat 的 139 万明显小了太多。这是怎么回事呢？难道是工具本身出了错吗？

别着急，在怀疑工具之前，我们再来回想一下，前面讲到的几种上下文切换场景。其中有一点提到，Linux 调度的基本单位实际上是线程，而我们的场景 sysbench 模拟的也是线程的调度问题，那么，是不是 pidstat 忽略了线程的数据呢？

通过运行 man pidstat，你会发现，pidstat 默认显示进程的指标数据，加上 -t 参数后，才会输出线程的指标。

所以，我们可以在第三个终端里，Ctrl+C 停止刚才的 pidstat 命令，再加上 -t 参数，重试一下看看：

```

1 # 每隔 1 秒输出一组数据（需要 Ctrl+C 才结束）
2 # -wt 参数表示输出线程的上下文切换指标
3 $ pidstat -wt 1
4 08:14:05      UID      TGID      TID    cswch/s nvcschw/s  Command
5 ...
6 08:14:05      0      10551      -      6.00      0.00  sysbench
7 08:14:05      0      -      10551      6.00      0.00  |__sysbench
8 08:14:05      0      -      10552  18911.00 103740.00  |__sysbench
9 08:14:05      0      -      10553  18915.00 100955.00  |__sysbench
10 08:14:05      0      -      10554  18827.00 103954.00  |__sysbench
11 ...

```

 复制代码

现在你就能看到了，虽然 sysbench 进程（也就是主线程）的上下文切换次数看起来并不多，但它的子线程的上下文切换次数却有很多。看来，上下文切换罪魁祸首，还是过多的 sysbench 线程。

我们已经找到了上下文切换次数增多的根源，那是不是到这儿就可以结束了呢？

当然不是。不知道你还记不记得，前面在观察系统指标时，除了上下文切换频率骤然升高，还有一个指标也有很大的变化。是的，正是中断次数。中断次数也上升到了 1 万，但到底是什么类型的中断上升了，现在还不清楚。我们接下来继续抽丝剥茧找源头。

既然是中断，我们都知道，它只发生在内核态，而 pidstat 只是一个进程的性能分析工具，并不提供任何关于中断的详细信息，怎样才能知道中断发生的类型呢？

没错，那就是从 /proc/interrupts 这个只读文件中读取。/proc 实际上是 Linux 的一个虚拟文件系统，用于内核空间与用户空间之间的通信。/proc/interrupts 就是这种通信机制的一部分，提供了一个只读的中断使用情况。

我们还是在第三个终端里，Ctrl+C 停止刚才的 pidstat 命令，然后运行下面的命令，观察中断的变化情况：

```
1 # -d 参数表示高亮显示变化的区域
2 $ watch -d cat /proc/interrupts
3           CPU0           CPU1
4 ...
5 RES:      2450431      5279697   Rescheduling interrupts
6 ...
```

[复制代码](#)

观察一段时间，你可以发现，变化速度最快的是**重调度中断**（RES），这个中断类型表示，唤醒空闲状态的 CPU 来调度新的任务运行。这是多处理器系统（SMP）中，调度器用来分散任务到不同 CPU 的机制，通常也被称为**处理器间中断**（Inter-Processor Interrupts, IPI）。

所以，这里的中断升高还是因为过多任务的调度问题，跟前面上下文切换次数的分析结果是一致的。

通过这个案例，你应该也发现了多工具、多方面指标对比观测的好处。如果最开始时，我们只用了 pidstat 观测，这些很严重的上下文切换线程，压根儿就发现不了了。

现在再回到最初的问题，每秒上下文切换多少次才算正常呢？

**这个数值其实取决于系统本身的 CPU 性能。**在我看来，如果系统的上下文切换次数比较稳定，那么从数百到一万以内，都应该算是正常的。但当上下文切换次数超过一万次，或者切换次数出现数量级的增长时，就很可能已经出现了性能问题。

这时，你还需要根据上下文切换的类型，再做具体分析。比方说：

- 自愿上下文切换变多了，说明进程都在等待资源，有可能发生了 I/O 等其他问题；
- 非自愿上下文切换变多了，说明进程都在被强制调度，也就是都在争抢 CPU，说明 CPU 的确成了瓶颈；



- 中断次数变多了，说明 CPU 被中断处理程序占用，还需要通过查看 `/proc/interrupts` 文件来分析具体的中断类型。

## 小结

今天，我通过一个 `sysbench` 的案例，给你讲了上下文切换问题的分析思路。碰到上下文切换次数过多的问题时，**我们可以借助 `vmstat`、`pidstat` 和 `/proc/interrupts` 等工具**，来辅助排查性能问题的根源。

## 思考

最后，我想请你一起来聊聊，你之前是怎么分析和排查上下文切换问题的。你可以结合这两节的内容和你自己的实际操作，来总结自己的思路。

欢迎在留言区和我讨论，也欢迎把这篇文章分享给你的同事、朋友。我们一起在实战中演练，在交流中学习。



©版权归极客邦科技所有，未经许可不得转载

上一篇 03 | 基础篇：经常说的 CPU 上下文切换是什么意思？（上）

下一篇 05 | 基础篇：某个应用的CPU使用率居然达到100%，我该怎么办？

写留言

精选留言



行者

👍 29



结合前两节，首先通过uptime查看系统负载，然后使用mpstat结合pidstat来初步判断到底是cpu计算量大还是进程争抢过大或者是io过多，接着使用vmstat分析切换次数，以及切换类型，来进一步判断到底是io过多导致问题还是进程争抢激烈导致问题。

2018-11-28

### 作者回复



2018-11-28



郝中阳

13

我用的centos, yum装的sysbench。执行后很快完事了了的可以设置下max-requests，默认max-requests是1w所以很快就结束了。

```
sysbench --num-threads=10 --max-time=300 --max-requests=10000000 --test=threads run
```

有的朋友/proc/interrupts时看不见RES是因为窗口开太小了RES在最下面。

2018-11-29

### 作者回复

👍 谢谢分享经验，其他用centos的同学可以参考一下

2018-11-30



CYH

5

老师请教一下：我的是centos7系统，执行sysbench后，watch -d cat /proc/interrupts并没有发现您文中描述的重调度中断指标呢？

2018-11-28



lucian

5

过多上下文切换会缩短进程运行时间 vmstat 1 1：分析内存使用情况、cpu上下文切换和中断的次数。cs每秒上下文切换的次数，in每秒中断的次数，r运行或等待cpu的进程数，b中断睡眠状态的进程数。 pidstat -w 5：查看每个进程详细情况。cswch（每秒自愿）上下文切换次数，如系统资源不足导致，nvcsch每秒非自愿上下文切换次数，如cpu时间片用完或高优先级线程 案例分析： sysbench：多线程的基准测试工具，模拟context switch 终端1： sysbench --threads=10 --max-time=300 threads run 终端2： vmstat 1： sys列占用84%说明主要被内核占用，ur占用16%；r就绪队列8；in中断处理1w，cs切换139w==>等待进程过多，频繁上下文切换，内核cpu占用率升高 终端3： pidstat -w -u 1： sysbench的cpu占用100%（-wt发现子线程切换过多），其他进程导致上下文切换 watch -d cat /proc/interrupts：查看另一个指标中断次数，在/proc/interrupts中读取，发现重调度中断res变化速度最快 总结：cswch过多说明资源IO问题，nvcsch过多说明调度争抢cpu过多，中断次数变多说明cpu被中断程序调用

2018-11-28

### 作者回复

总结的好👍

2018-11-28



姜小鱼

5



老师 为什么我执行sysbench之后很快就结束了? sysbench --num-threads=10 --max-time=600 --test=threads run 我用的是ubuntu16

2018-11-28

### 作者回复

有没有报错误? 可以加上一debug看看有没有错误消息

2018-11-28



唐瑞甫

👍 3

发现一个不太严谨的地方, 即使没有开sysbench, 用watch -d /proc/interrupts的时候 RES的变化也是最大的, 这个时候in跟cs都不高

2018-11-28

### 作者回复

中断次数是多少?

2018-11-28



echo

👍 3

老师你好, 有个难题希望能指导下排查

我有个系统, 跑的是32核48G的云主机, load经常超过CPU核数, 峰值时load5可达到CPU核数的3倍, 但是CPU利用率不超过50%左右。

其他关键数据: I/O wait 不超过0.1, 网络流量没超出网卡QOS, R状态的进程数也就一两个, 没有D状态的进程。系统只要跑一个CPU密集型的Java进程, 线程数2-3k。另外load、CPU、网卡流量的曲线是一致的。

通读了你的第二篇文章, 按文章指导能排查的都排查了, 接下来应该从哪方面着手定位load高的根因呢?

2018-11-28

### 作者回复

CPU使用率都是哪种高? 具体到每个线程, 又是哪种CPU使用率高? 先找出最高的类型再继续分析

2018-11-28



Haku

👍 3

Ubuntu16.04LTS下:

# 以 10 个线程运行 5 分钟的基准测试, 模拟多线程切换的问题

\$ sysbench --num-threads=10 --max-time=300 --test=threads run

2018-11-28

### 作者回复

赞👍

2018-11-28



高峰

👍 3

Pidstats确实是把利器啊

2018-11-28



Linuxer

👍 2

我们之前是如果系统CPU不高根本不会去关注上下文切换，但是这种情况下以前也观测到cs有几十万的情况，所以我想请教一个问题，什么情况下需要关注上下文切换呢？

2018-11-28

### 作者回复

是的，cs值不是绝对的，所以最好是监控起来，看变化情况，比如是不是数量级的增长

2018-11-28



TERRY.ROD

👍 1

1. stress和sysbench两个工具在压测过程中的对比发现：

stress基于多进程的，会fork多个进程，导致进程上下文切换，导致us开销很高；

sysbench基于多线程的，会创建多个线程，单一进程基于内核线程切换，导致sy的内核开销很高；

具体可以通过vmstat对比

```
stress -c 8 -i 16 -t 600
```

```
vmstat 1 5
```

```
sysbench --threads=20 --time=300 threads run
```

```
vmstat 1 5
```

2. 和鸟哥说的一样，不懂多看man，see also的命令基本涵盖了老师讲解的

3. 建议结合操作系统完成后，再看这块教程，整体会更系统性，看问题会更加客观

希望对大家有帮助

2018-12-03



发条橙子。

👍 1

案例分析：

登录到服务器，现在系统负载怎么样。高的话有三种情况，首先是cpu使用率，其次是io使用率，之后就是两者都高。

cpu 使用率高，可能确实是使用率高，也的可能实际处理不高而是进程太多切换上下文频繁，也可能是进程内线程的上下文切换频繁

io 使用率高，说明 io 请求比较大，可能是 文件io、网络io。

工具：

系统负载：uptime（watch -d uptime）看三个阶段平均负载

系统整体情况：mpstat（mpstat -p ALL 3）查看每个cpu当前的整体状况，可以重点看用户态、内核态、以及io等待三个参数

系统整体的平均上下文切换情况：vmstat（vmstat 3）可以重点看r（进行或等待进行的进程）、b（不可中断进程/io进程）、in（中断次数）、cs（上下文切换次数）

查看详细的上下文切换情况：pidstat (pidstat -w(进程切换指标)/-u (cpu使用指标) /-wt(线程上下文切换指标)) 注意看是自愿上下文切换、还是被动上下文切换  
io使用情况：iostat

模拟场景工具：

stress：模拟进程、io

sysbench：模拟线程数

2018-12-02

作者回复

总结的很好，继续保持👍

2018-12-03



冯宇

👍 1

友情提醒，sudo -i就可以快速切换到root啦😊 不加-i的话是以非登录模式切换，不会拿到root的环境变量

2018-11-30



茴香根

👍 1

打卡本节课程，在使用Linux一些监控命令行时候常常碰到列宽和下面的数据错位的情况，比如数据过大，占了两列，导致数据错位，不方便观察，不知老师可有好的工具或方法解决。

2018-11-28

作者回复

这是常见的问题，一般用宽显示器会好些

2018-11-28



C家族铁粉儿

👍 1

『D5打卡』

不用root权限的Linux用户，不是好的用户😊

这几天访问/proc 只读文件的次数，比以前几个月都多，老实说，学会pidstat、vmstat这些工具的靠谱使用方法，就值了。不过还是要记住，工具不是全部  
乌班图真的稳，跟着老师操作，基本没啥问题

2018-11-28

作者回复



2018-11-28



maoxiajun

👍 0

lesson4打卡

2018-12-11

作者回复



2018-12-11



汤圆

0

```
procs -----memory----- ---swap-- -----io----- -system-- -----cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
1 0 520 3831600 116 7871856 0 0 0 11 0 0 0 0 99 0 0
0 0 520 3831264 116 7871868 0 0 0 0 2285 2400 0 0 100 0 0
0 0 520 3830956 116 7871924 0 0 0 11 2591 2818 2 0 98 0 0
0 0 520 3830268 116 7871972 0 0 0 64 2440 2673 1 0 99 0 0
```

在使用vmstat 5 查看系统性能的时候，第一行cs, us小，之后数值都很大，这是为什么？

2018-12-10

### 作者回复

这是vmstat的一个坑，第一行是系统启动以来的平均值，直接忽略即可。后面的各行才是你设置的间隔时间的平均值

2018-12-11



如果

DAY4,打卡

0

2018-12-10

### 作者回复

+1赞

2018-12-10



力挽狂澜爆炸输出的臭臭宁

0

为什么/proc/interrupts显示的RES中断数有245万多，而vmstat显示的只有1万？

2018-12-07

### 作者回复

第一个是累加值，第二个是vmstat采集时间的差值

2018-12-07



Geng

0

```
15: 432533 IO-APIC 15-edge ata_piix
24: 0 PCI-MSI 65536-edge virtio1-config
25: 5881576 PCI-MSI 65537-edge virtio1-req.0
26: 0 PCI-MSI 49152-edge virtio0-config
27: 16526409 PCI-MSI 49153-edge virtio0-input.0
28: 222 PCI-MSI 49154-edge virtio0-output.0
NMI: 0 Non-maskable interrupts
LOC: 39324969 Local timer interrupts
SPU: 0 Spurious interrupts
MCE: 0 Machine check exceptions
MCP: 1352 Machine check polls
```

HYP: 0 Hypervisor callback interrupts

我的设备上显示是LOC较高

2018-12-06